C introduction

The C standard library

Richard Mörbitz, Manuel Thieme

Contents

Don't reinvent the wheel

If you split your problem into sub-problems and solve each of them in a seperate function, you're on a good way.

However, many of the very basic sub-problems have already been solved ages ago.

These solution are provided in *libraries* such as the *glibc* used by the *gcc*.

Library implemetations are safer and more efficient than yours will ever be, *plus* you save a lot of time using them.

The Hitchhiker's Guide to the standard library

The functions are *declared* in a *header file*.
Each header file has a certain name and the file extension *.h*.

The *include* preprocessor statement puts them into your program, e.g.

```c
#include <stdio.h>  /* We have done that so many times */
```

The actual function implementation is linked dynamically to your program during runtime. Let's not care about that.

With less than 30 header files, the C library is rather small.
We will go through the ones that might be the most useful to you.

assert.h

- Contains the *assert()* macro, witch evaluates the truth value of an expression
- If it's true, nothing happens
- Else the program aborts and an error message is printed

→ useful to avoid undefined behaviour / worse errors at runtime

We can also use it if we just want to test things:

```
unisgned int input;
printf("Enter a one-digit decimal number:\n");
scanf("%d", &input);
assert(input < 10);
```

math.h

- Declares a lot of mathematic functions
- Finally you're able to calculate square roots, logarithms, etc.
- Most of those functions have *double* arguments and return values

If you use functions from *math.h*, add the *-lm* as the **last** option to *gcc* to avoid errors:

```
gcc main.c -lm
```

stdio.h

- Declares the basic functions to read and write data
- You know *printf()* and *scanf()*, but there is more:
- Characters, unprocessed and formatted strings
- Command line I/O and file access
- Many functions for high-level file management

As an example, *puts()* can be used instead of *printf()* if you have a basic string without placeholders - '\n' is added automatically:

```
puts("Hello World!");
/* Equivalent to printf("Hello World!\n") */
```

stdlib.h

This probably is the most powerful header providing various different functionalities. Here is just an excerpt:

- *EXIT_SUCCESS* and *EXIT_FAILURE* constants as an alternative to returning *0* or *1* at the end of *main()*
- Alternative ways to exit the program
- Generation of pseudo-random numbers
- Search and sorting function
- Dynamic memory management

. . . and more things you haven't even heard of

string.h

string.h

Wait! Strings?

string.h

Wait! Strings?

Yes, there are strings in C.
They are just handled differently from what you would expect.

string.h is crucial if you want to work with C strings seriously.
We will use some of the functions declared there in later lessons.

time.h

- Data types to store different time formats
- Functions to get the calendar and cpu time
- Functions to format time values
- Functions to measure and calculate time differences

Handling time usually is quite complicated, but with the help of *time.h* it gets a lot easier.

Measure the execution time of your programs to see how efficient they are!

Documentation

Learning all the library functions is way less effective than knowing where to look them up quickly.
Man page is a Unix tool containing documentation of programs, system calls and libraries - such as the C standard library.

To access a certain man page, just type:

```
$ man page
```

Example for *printf()*:

```
$ man printf
```

However, this describes the shell command *printf*.

Effective use of *man*

Man has many sections, library functions are in #3.
Write the section number between *man* and the page:

```
$ man 3 printf
```

To get all pages *printf* occurs in, use the *-k* option:

```
$ man -k printf
```

If you need more information on *man* - it has its own man page:

```
$ man man
```

Pretty output

To solve the following tasks, you need to use some functions that may be new to you.
No explanation is given on them, use *man* to find out how they work.

- Write a program that outputs a function table like the following:
- In the first row, print all values x from 1 to 10. In the second row, print $1/x$ from 1 to 10.
- Play around with the format string of *printf()* to create a nice table-like output.
- **Experts**: Print a multiplication table instead (from $1 * 1$ to $10 * 10$)

Aleia iacta est

- Write a program that simulates a dice using the *rand()* function.
- **Experts**: Simulate "real" randomness that cannot be recreated.

Aleia iacta est

- Write a program that simulates a dice using the *rand()* function.
  - Hint: have a look at *srand()* as well.
- **Experts**: Simulate "real" randomness that cannot be recreated.

Aleia iacta est

- Write a program that simulates a dice using the *rand()* function.
    - Hint: have a look at *srand()* as well.
- **Experts**: Simulate "real" randomness that cannot be recreated.
    - Hint: choose the *seed* for *srand()* dynamically.

Aleia iacta est

- Write a program that simulates a dice using the *rand()* function.
  - Hint: have a look at *srand()* as well.
- **Experts**: Simulate "real" randomness that cannot be recreated.
  - Hint: choose the *seed* for *srand()* dynamically.
  - Hint: what value is different every*time* you start the program?

Math exam

- Write a program that asks the user to solve some simple mathematic tasks.
- If the user answers wrong, the program should abort with an error.

- **Experts**: At the end, print the time it took the user to answer all the questions.

Math exam

- Write a program that asks the user to solve some simple mathematic tasks.
- If the user answers wrong, the program should abort with an error.
  - Remember the *assert()* macro?
- **Experts**: At the end, print the time it took the user to answer all the questions.